(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 March 2001 (08.03.2001)

PCT

(10) International Publication Number
WO 01/16737 A2

(51) International Patent Classification[7]: G06F 9/50

(21) International Application Number: PCT/US00/24147

(22) International Filing Date: 31 August 2000 (31.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/152,151	31 August 1999 (31.08.1999)	US
60/220,748	26 July 2000 (26.07.2000)	US
60/220,794	26 July 2000 (26.07.2000)	US

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier applications:

US	60/152,151 (CIP)
Filed on	31 August 1999 (31.08.1999)
US	60/220,748 (CIP)
Filed on	26 July 2000 (26.07.2000)
US	60/220,794 (CIP)
Filed on	26 July 2000 (26.07.2000)

(71) Applicant *(for all designated States except US)*: **TIMES N SYSTEMS, INC.** [US/US]; Bldg. B, Suite P, 1908 Kramer Lane, Austin, TX 78758 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **WEST, Lynn, Parker** [US/US]; 10201 Pantera Drive, Austin, TX 78759 (US). **SCARDAMALIA, Ted** [US/US]; 1910 Clearwater Dr., Round Rock, TX 78681 (US).

(74) Agent: **BRUCKNER, John, J.**; Wilson, Sonsini, Goodrich & Rosati, 650 Page Mill Road, Palo Alto, CA 94304-1050 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 01/16737 A2

(54) Title: CACHE-COHERENT SHARED-MEMORY CLUSTER

(57) Abstract: Methods, systems and devices are described for a cache-coherent shared-memory cluster. A system includes a multiplicity of workstations and a shared memory unit (SMU) interconnected and arranged such that memory accesses by a given workstation in a set of address ranges will be to its local, private memory and that memory accesses to a second set of address ranges will be to shared memory and arranged such that accesses to the shared memory unit are recorded and signaled by a cache-emulator adapter (CEA) capable of recognizing and responding to cache-coherence signals within the given workstation. The methods, systems and devices provide advantages because the speed and scalability of parallel processor systems is enhanced.

CACHE-COHERENT SHARED-MEMORY CLUSTER

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to the field of computer systems based on multiple processors and shared memory. More particularly, the invention relates to computer systems that utilize a cache-coherent shared-memory cluster.

2. Discussion of the Related Art

The clustering of workstations is a well-known art. In the most common cases, the clustering involves workstations that operate almost totally independently, utilizing the network only to share such services as a printer, license-limited applications, or shared files.

In more-closely-coupled environments, some software packages (such as NQS) allow a cluster of workstations to share work. In such cases the work arrives, typically as batch jobs, at an entry point to the cluster where it is queued and dispatched to the workstations on the basis of load.

In both of these cases, and all other known cases of clustering, the operating system and cluster subsystem are built around the concept of message-passing. The term message-passing means that a given workstation operates on some portion of a job until communications (to send or receive data, typically) with another workstation is necessary. Then, the first workstation prepares and communicates with the other workstation.

Another well-known art is that of clustering processors within a machine, usually called a Massively Parallel Processor or MPP, in which the techniques are essentially identical to those of clustered workstations. Usually, the bandwidth and latency of the interconnect network of an MPP are more highly optimized, but the system operation is the same.

In the general case, the passing of a message is an extremely expensive operation; expensive in the sense that many CPU cycles in the sender and receiver are consumed by the process of sending, receiving, bracketing,

verifying, and routing the message, CPU cycles that are therefore not available for other operations. A highly streamlined message-passing subsystem can typically require 10,000 to 20,000 CPU cycles or more.

There are specific cases wherein the passing of a message requires significantly less overhead. However, none of these specific cases is adaptable to a general-purpose computer system.

Message-passing parallel processor systems have been offered commercially for years but have failed to capture significant market share because of poor performance and difficulty of programming for typical parallel applications. Message-passing parallel processor systems do have some advantages. In particular, because they share no resources, message-passing parallel processor systems are easier to provide with high-availability features. What is needed is a better approach to parallel processor systems.

There are alternatives to the passing of messages for closely-coupled cluster work. One such alternative is the use of shared memory for inter-processor communication.

Shared-memory systems, have been much more successful at capturing market share than message-passing systems because of the dramatically superior performance of shared-memory systems, up to about four-processor systems. In Search of Clusters, Gregory F. Pfister 2nd ed. (January 1998) Prentice Hall Computer Books, ISBN: 0138997098 describes a computing system with multiple processing nodes in which each processing node is provided with private, local memory and also has access to a range of memory which is shared with other processing nodes. The disclosure of this publication in its entirety is hereby expressly incorporated herein by reference for the purpose of indicating the background of the invention and illustrating the state of the art.

However, providing high availability for traditional shared-memory systems has proved to be an elusive goal. The nature of these systems, which share all code and all data, including that data which controls the shared operating systems, is incompatible with the separation normally required for

high availability. What is needed is an approach to shared-memory systems that improves availability.

Although the use of shared memory for inter-processor communication is a well-known art, prior to the teachings of U.S. Ser. No. 09/273,430, filed March 19, 1999, entitled Shared Memory Apparatus and Method for Multiprocessing Systems, the processors shared a single copy of the operating system. The problem with such systems is that they cannot be efficiently scaled beyond four to eight way systems except in unusual circumstances. All known cases of said unusual circumstances are such that the systems are not good price-performance systems for general-purpose computing.

The entire contents of U.S. Patent Applications 09/273,430, filed March 19, 1999 and PCT/US00/01262, filed January 18, 2000 are hereby expressly incorporated by reference herein for all purposes. U.S. Ser. No. 09/273,430, improved upon the concept of shared memory by teaching the concept which will herein be referred to as a tight cluster. The concept of a tight cluster is that of individual computers, each with its own CPU(s), memory, I/O, and operating system, but for which collection of computers there is a portion of memory which is shared by all the computers and via which they can exchange information. U.S. Ser. No. 09/273,430 describes a system in which each processing node is provided with its own private copy of an operating system and in which the connection to shared memory is via a standard bus. The advantage of a tight cluster in comparison to an SMP is "scalability" which means that a much larger number of computers can be attached together via a tight cluster than an SMP with little loss of processing efficiency.

What is needed are improvements to the concept of the tight cluster. What is also needed is an expansion of the concept of the tight cluster.

Another well-known art is the use of memory caches to improve performance. Caches provide such a significant performance boost that most modern computers use them. At the very top of the performance (and price) range all of memory is constructed using cache-memory technologies. However, this is such an expensive approach that few manufacturers use it. All manufacturers of personal computers (PCs) and workstations use caches except

for the very low end of the PC business where caches are omitted for price reasons and performance is, therefore, poor.

Caches, however, present a problem for shared-memory computing systems; the problem of coherence. As a particular processor reads or writes a word of shared memory, that word and usually a number of surrounding words are transferred to that particular processor's cache memory transparently by cache-memory hardware. That word and the surrounding words (if any) are transferred into a portion of the particular processor's cache memory that is called a cache line or cache block.

If the transferred cache line is modified by the particular processor, the representation in the cache memory will become different from the value in shared memory. That cache line within that particular processor's cache memory is, at that point, called a "dirty" line. The particular processor with the dirty line, when accessing that memory address will see the new (modified) value. Other processors, accessing that memory address will see the old (unmodified) value in shared memory. This lack of coherence between such accesses will lead to incorrect results.

Modern computers, workstations, and PCs which provide for multiple processors and shared memory, therefore, also provide high-speed, transparent cache coherence hardware to assure that if a line in one cache changes and another processor subsequently accesses a value which is in that address range, the new values will be transferred back to memory or at least to the requesting processor.

Caches can be maintained coherent by software provided that sufficient cache-management instructions are provided by the manufacturer. However, in many cases, an adequate arsenal of such instructions are not provided. Moreover, even in cases where the instruction set is adequate, the software overhead is so great that no examples of are known of commercially successful machines which use software-managed coherence.

Thus, the existing hardware and software cache coherency approaches are unsatisfactory. What is also needed, therefore, is a better approach to cache coherency.

SUMMARY OF THE INVENTION

A goal of the invention is to simultaneously satisfy the above-discussed requirements of improving and expanding the tight cluster concept which, in the case of the prior art, are not satisfied.

One embodiment of the invention is based on an apparatus, comprising: a central shared memory unit; a first processing node coupled to said central shared memory unit via a first interconnection; and a second processing node coupled to said central shared memory unit via a second interconnection. Another embodiment of the invention is based on a method, comprising: recording memory accesses by a workstation to a shared memory unit (SMU) by a cache emulator adapter (CEA) capable of recognizing and responding to cache-coherence signals within the workstation. Another embodiment of the invention is based on an electronic media, comprising: a computer program adapted to record memory accesses by a workstation to a shared memory unit (SMU) by a cache emulator adapter (CEA) capable of recognizing and responding to cache-coherence signals within the workstation. Another embodiment of the invention is based on a system, comprising a multiplicity of workstations and a shared memory unit (SMU) interconnected and arranged such that memory accesses by a given workstation in a set of address ranges will be to its local, private memory and that memory accesses to a second set of address ranges will be to shared memory and arranged such that accesses to the shared memory unit are recorded and signaled by a cache emulator adapter (CEA) capable of recognizing and responding to cache-coherence signals within the given workstation. Another embodiment of the invention is based on a system, comprising: multiple processing nodes and one or more shared memory nodes; each processing node containing some memory which is local and private to that node, and each shared memory node containing some other memory which is visible and usable by all nodes; and a local cache system which provides caching for a first memory address region, and one or more shared-memory cache systems which provide caching for one or more other address regions. Another embodiment of the invention is based on a system,

comprising multiple processing nodes; each processing node containing some memory which is local and private to that node, and a portion of other memory which is visible and usable by all nodes.

These, and other goals and embodiments of the invention will be better appreciated and understood when considered in conjunction with the following description and the accompanying drawings. It should be understood, however, that the following description, while indicating preferred embodiments of the invention and numerous specific details thereof, is given by way of illustration and not of limitation. Many changes and modifications may be made within the scope of the invention without departing from the spirit thereof, and the invention includes all such modifications.

BRIEF DESCRIPTION OF THE DRAWINGS

A clear conception of the advantages and features constituting the invention, and of the components and operation of model systems provided with the invention, will become more readily apparent by referring to the exemplary, and therefore nonlimiting, embodiments illustrated in the drawings accompanying and forming a part of this specification, wherein like reference characters (if they occur in more than one view) designate the same parts. It should be noted that the features illustrated in the drawings are not necessarily drawn to scale.

FIG. 1 illustrates a block schematic view of a share-as-needed system, representing an embodiment of the invention.

FIG. 2 illustrates a block schematic view of a cache and directory for the system of FIG. 1.

FIG. 3 illustrates a block schematic view of another cache and directory for the system of FIG. 1.

FIG. 4 illustrates a block schematic view of a share-as-needed system with distributed, shared memory, representing an embodiment of the invention.

FIG. 5 illustrates a block schematic view of a share-as-needed system distributed memory node, representing an embodiment of the invention.

DESCRIPTION OF PREFERRED EMBODIMENTS

The invention and the various features and advantageous details thereof are explained more fully with reference to the nonlimiting embodiments that are illustrated in the accompanying drawings and detailed in the following description of preferred embodiments. Descriptions of well known components and processing techniques are omitted so as not to unnecessarily obscure the invention in detail.

The teachings of U.S. Ser. No. 09/273,430 include a system which is a single entity; one large supercomputer. The invention is also applicable to a cluster of workstations, or even a network.

The invention is applicable to systems of the type of Pfister or the type of U.S. Ser. No. 09/273,430 in which each processing node has its own copy of an operating system. The invention is also applicable to other types of multiple processing node systems.

The context of the invention can include a tight cluster as described in U.S. Ser. No. 09/273,430. A tight cluster is defined as a cluster of workstations or an arrangement within a single, multiple-processor machine in which the processors are connected by a high-speed, low-latency interconnection, and in which some but not all memory is shared among the processors. Within the scope of a given processor, accesses to a first set of ranges of memory addresses will be to local, private memory but accesses to a second set of memory address ranges will be to shared memory. The significant advantage to a tight cluster in comparison to a message-passing cluster is that, assuming the environment has been appropriately established, the exchange of information involves a single STORE instruction by the sending processor and a subsequent single LOAD instruction by the receiving processor.

The establishment of the environment, taught by U.S. Ser. No. 09/273,430 and more fully by companion disclosures (U.S. Provisional Application Ser. No. 60/220,794, filed July 26, 2000; U.S. Provisional Application Ser. No. 60/220,748, filed July 26, 2000; WSGR 15245-712; WSGR 15245-713; WSGR 15245-715; WSGR 15245-716; WSGR 15245-717;

WSGR 15245-718; WSGR 15245-719; and WSGR 15245-720, the entire contents of all which are hereby expressly incorporated herein by reference for all purposes) can be performed in such a way as to require relatively little system overhead, and to be done once for many, many information exchanges. Therefore, a comparison of 10,000 instructions for message-passing to a pair of instructions for tight-clustering, is valid.

The invention can be embodied as a system in which the shared memory is distributed within the workstations of the cluster. The shared memory can be distributed evenly or unevenly.

The invention can include a computer system in which there are multiple processing nodes, for which each node is provided with some local (private) memory and for which some memory is provided which is visible to all processing nodes. The invention can include means for keeping the shared memory coherent with caches in the processing nodes.

In U.S. Ser. No. 09/273,430, a system is described in which a multiplicity of processing nodes, each with private, local memory, have equal access to a shared pool of memory via a standard bus. In that application, the preferred embodiment utilizes PCI adapters, resident on the PCI bus of each processing node for means to access the shared memory. This invention teaches multiple mechanisms based on that system (herein called a Processor Team) for keeping the caches within the processing nodes coherent with the shared memory.

The invention can be described in terms of a set-associative cache using the AMESI protocol. The AMESI protocol, for reference, stands for Absent, Modified, Exclusive, Shared and Invalid. Absent and Invalid are reported differently by the cache upon interrogation, but the action taken is the same, so the description hereafter will refer only to MESI.

In a MESI system, if a cache interrogation results in Invalid, then the cache does not have the data and the interrogator reacts appropriate to that response. If a cache interrogation results in Exclusive, then the cache has the only valid copy of the data and the interrogator reacts appropriately to that response. If a cache interrogation results in Shared, then the cache has one valid

Such a card would not in reality be a memory card, but would be responsive to accesses (LOADS and STORES) across a range of memory addresses, said range being a range designed a priori to be the range in which shared memory resides. Within said range, the memory card would translate accesses into requests and responses to the shared memory unit where the actual shared data would be stored into and retrieved from. Such a memory-alias card or adapter is of value and workable only if there is no cache or if software-managed cache coherence were used.

The present invention describes a different adapter within the workstation/PC which not only provides an interface to the interconnection to the SMU but which also provides hardware cache coherence for accesses to shared memory. Most PCs and workstations which provide for multiple processors (SMPs) provide a bus to which each data cache has access via which each "snoops" the bus activity of other processors. If a processor requires a memory access which may involve cache coherence issues, that access is first pre-signaled on the snooped bus. Any other cache controller for which the operation may cause loss of coherence signals such potential loss to the originating source and operations which preserve coherence then are executed. This invention relates to such PCs and workstations.

There are many such coherence maintenance schemes and this invention does not teach nor describe in detail such schemes; they are well known to those skilled in the art. The schemes involve some kind of positive response (ACK) if cache coherence actions are required, and negative response (NAK) if the particular cache has no requirement for coherence actions. The present invention can be applied to any such scheme and for each will be designed to use the protocols of the particular scheme in use by the workstation.

The invention can include an interface within the PC or workstation which emulates a cache including cache-coherence control mechanisms, and fully emulates those data and control signals on the snooped bus, but which is not a cache nor is it the usual SMP companion processor found behind such caches, but rather is an interconnect interface to the communication link to the

SMU. The invented adapter is hereafter called the cache emulation adapter (CEA).

In a system having some private, local memory in each of a multiplicity of processing nodes (PRNs) and some shared memory in one or more shared-memory nodes (SMNs), the invention can include assuring that caches within the processing nodes are kept coherent with the shared memory. The system within which the invention operates can include a multiplicity of workstations (which may be PCs) and including a shared memory unit (SMU) in which the addressable scope of memory within each workstation is subdivided such that one, or more address ranges, are local memory within each workstation which is/are not addressable by other workstations and one, or more, other address ranges the memory which is shared by all workstations.

In one embodiment of the invention, the SMU provides means for signaling to each CEA when a shared-memory cache line has been accessed by any CEA. All CEAs therefore keep a hashed, partial directory indicating which shared-memory cache lines may need coherence operations. For most memory accesses to memory, the originating CPU can determine coherence status within its own cache and cache controls so no external snooping action is required. When cache coherence (snooping action) is required, and if the address range is to private memory, the CEA will respond with the NAK response. For shared-memory accesses by the CPU (or CPUs) within a given workstation which require potential coherence action, the CEA will detect the request and will hold off (some protocols require it to signal HOLD OFF to the CPU, which it will do if appropriate) and signal the SMU which will, in turn, flush or invalidate the cache line in the owning cache (if any) and send the cache line to the requesting workstation.

In another embodiment, the CEA may not be provided with a hashed, partial directory. In this case, all snoops to global memory will require CEA-SMU transactions to invalidate the needed cache line or to determine that it is clean.

In another embodiment, each CEA can be provided with a full directory of all shared lines cached in all workstations and can obtain the required cache

line directly, notifying other CEAs to update ownership information for the cache line.

For any of the embodiments above, the CEA can be provided with caching to improve speed of response to shared memory at any workstation. The cache would be much larger than the typical CPU cache and would cache much larger blocks: memory pages, for example.

In another embodiment of the invention, the shared memory is accessed in units hereafter referred to as pages. The operating system in use is then configured to mark the pages as non-cacheable. Then, when a particular PRN accesses a shared page, the data from that page is retrieved from shared memory, and, being non-cacheable, is not placed into the cache of the PRN. Similarly, when data is written to that page, it is not written into the cache of that PRN by virtue of being marked non-cacheable. Therefore, any other processor subsequently accessing that data will see the same data, achieving the goal of keeping all caches coherent with respect to shared data.

In another embodiment, which can improve overall system performance for certain data access patterns, the same methodology is used, but in the memory access means is included to provide a shared memory cache of the data more local (less latency) than the shared memory.

Figure 1 shows such a memory access means, and shows the processing nodes and shared-memory node for a share-as-needed embodiment. A plurality of processing nodes 101 are coupled to a shared-memory node 102 with processing node to shared-memory node links 103.

Figure 2 shows a portion of a system similar to the system of figure 1. Figure 2 shows a processing node 201 coupled to a shared-memory node 202 with a processing node to shared-memory node line 203. The processing node 201 includes a cache 204 for the caching of shared memory data. This cache 204 is not the same cache as is used for local memory, but rather is kept separate. The normal local-memory cache can be present without skirting the teachings of this invention.

In figure 2, the processing node 201 also includes a directory 205. The directory 205 is the directory indicating the contents of cache 205. The

preferred embodiment of this invention uses the MESI protocol and set-associative cache structure. Of course, the invention will work with other protocols and/or structures.

Figure 3 shows a preferred embodiment of this invention. In this embodiment, each processing node 301 is provided with a cache 304 for shared-memory accesses. A single directory 305 is provided at the shared memory node 302 for resolving cache coherence issues. The processing node 301 is coupled to the shared-memory node 302 with a processing node to shared-memory node link 303. In this preferred embodiment, if the processing node 301 accesses a shared-memory location and the cache 304 at that node contains the data in a state compatible with the type of access, the access is resolved at the processing node 301 and operation continues. If the cache 304 at that node does not contain the data or if it is in a state incompatible with the access, then a request agent is transferred to directory 305. Directory 305 then updates its entry for that cache line and takes the other action consistent with the original access type at the original processing node 301.

These consistent actions are common to directory-based coherence schemes as taught in the prior art, and include actions such as invalidating entries in other caches, changing the state of entries in other caches from "Exclusive" to "Shared" or removing Modified data from one cache and delivering that data to another cache.

The invention includes the concept of a "parallel" cache, independent of another cache. In the prior art, level-1 caches are taught, as are level-1 and level-2 combinations, and higher levels. For all known forms of these, the data in any given cache level is a superset or a subset of the data in lower-level caches.

The invention can include a parallel cache system. The first cache system may consist of a level-1 cache only, or of several levels of cache. The second cache system in this invention has data orthogonal to the first cache system. In the preferred embodiment, the first cache system caches data in a first address range, whereas the second cache system caches data in a second

address range. There may be more than a first and second cache parallel cache system provided, each responsive to a separate address range.

Figure 4 shows a share-as-needed system in which that portion of memory which is shared is distributed across a multiplicity of nodes rather than residing in a single, separate shared-memory node. A plurality of nodes 401 are interconnected with a node-node line 402.

Figure 5 shows a representative node that is similar to the nodes shown in figure 4. In figure 5, a processing node 501 is provided with local memory 502 and with shared memory 506. In addition, the node 501 is provided with a processor 504, a local-memory cache 503 and a shared-memory cache and directory 505. The node 501 also includes a shared-memory partition 506, a shared-memory interconnect 507 and shared-memory links 508.

The caches operate as described previously. The directory 505 is actually a portion of the system-wide shared-memory directory and communicates to other portions of said directory which are distributed within the other nodes. Such a distributed directory is taught in the prior art.

NUMA machines teach the concept of distributed memory. In NUMA machines, however, all memory is accessible by all processors in the system, and there is a single copy of the operating system running across all processors.

The invention can include a system comprising some memory which is private and local to the node, and on which is running a local copy of the operating system; and which includes some shared memory, visible to all processors, and, in such a system, this invention teaches the concept of distributing the shared memory across the nodes.

While not being limited to any particular performance indicator or diagnostic identifier, preferred embodiments of the invention can be identified one at a time by testing for the substantially highest performance. The test for the substantially highest performance can be carried out without undue experimentation by the use of a simple and conventional benchmark (speed) experiment.

The term substantially, as used herein, is defined as at least approaching a given state (e.g., preferably within 10% of, more preferably within 1% of, and

most preferably within 0.1% of). The term coupled, as used herein, is defined as connected, although not necessarily directly, and not necessarily mechanically. The term means, as used herein, is defined as hardware, firmware and/or software for achieving a result. The term program or phrase computer program, as used herein, is defined as a sequence of instructions designed for execution on a computer system. A program may include a subroutine, a function, a procedure, an object method, an object implementation, an executable application, an applet, a servlet, a source code, an object code, and/or other sequence of instructions designed for execution on a computer system.

Practical Applications of the Invention

A practical application of the invention that has value within the technological arts is waveform transformation. Further, the invention is useful in conjunction with data input and transformation (such as are used for the purpose of speech recognition), or in conjunction with transforming the appearance of a display (such as are used for the purpose of video games), or the like. There are virtually innumerable uses for the invention, all of which need not be detailed here.

Advantages of the Invention

A system, representing an embodiment of the invention, can be cost effective and advantageous for at least the following reasons. The invention improves the speed of parallel computing systems. The invention improves the scalability of parallel computing systems.

All the disclosed embodiments of the invention described herein can be realized and practiced without undue experimentation. Although the best mode of carrying out the invention contemplated by the inventors is disclosed above, practice of the invention is not limited thereto. Accordingly, it will be appreciated by those skilled in the art that the invention may be practiced otherwise than as specifically described herein.

For example, although the cache-coherent shared-memory cluster described herein can be a separate module, it will be manifest that the cache-coherent shared-memory cluster may be integrated into the system with which it

is associated. Furthermore, all the disclosed elements and features of each disclosed embodiment can be combined with, or substituted for, the disclosed elements and features of every other disclosed embodiment except where such elements or features are mutually exclusive.

It will be manifest that various additions, modifications and rearrangements of the features of the invention may be made without deviating from the spirit and scope of the underlying inventive concept. It is intended that the scope of the invention as defined by the appended claims and their equivalents cover all such additions, modifications, and rearrangements.

The appended claims are not to be interpreted as including means-plus-function limitations, unless such a limitation is explicitly recited in a given claim using the phrase "means for." Expedient embodiments of the invention are differentiated by the appended subclaims.

CLAIMS

What is claimed is:

1. An apparatus, comprising:

a central shared memory unit;

a first processing node coupled to said central shared memory unit via a first interconnection; and

a second processing node coupled to said central shared memory unit via a second interconnection.

2. The apparatus of claim 1, wherein said first interconnection includes a first interface and said second interconnection includes a second interface.

3. The apparatus of claim 2, wherein said first interface includes a first cache emulation adapter and said second interconnection includes a second cache emulation adapter.

4. A computer system, comprising the apparatus of claim 1.

5. An apparatus, comprising a first cache system and a second cache system, wherein the second cache system has data orthogonal to the first cache system.

6. A computer system, comprising the apparatus of claim 5.

7. A method, comprising:

recording memory accesses by a workstation to a shared memory unit (SMU) by a cache emulator adapter (CEA) capable of recognizing and responding to cache-coherence signals within the workstation.

8. The method of claim 7, further comprising signaling to another CEA when a shared-memory cache line has been accessed.

9. The method of claim 8, further comprising invalidating a cache line by a CEA-SMU transaction.

10. The method of claim 8, further comprising determining that a cache line is clean by a CEA-SMU transaction.

11. An electronic media, comprising: a computer program adapted to record memory accesses by a workstation to a shared memory unit (SMU) by a cache emulator adapter (CEA) capable of recognizing and responding to cache-coherence signals within the workstation.

12. A computer program comprising computer program means adapted to perform the step of recording memory accesses by a workstation to a shared memory unit (SMU) by a cache emulator adapter (CEA) capable of recognizing and responding to cache-coherence signals within the workstation when said computer program is run on a computer.

13. A computer program as claimed in claim 12, embodied on a computer-readable medium.

14. A system, comprising a multiplicity of workstations and a shared memory unit (SMU) interconnected and arranged such that memory accesses by a given workstation in a set of address ranges will be to its local, private memory and that memory accesses to a second set of address ranges will be to shared memory and arranged such that accesses to the shared memory unit are recorded and signaled by a cache emulator adapter (CEA) capable of recognizing and responding to cache-coherence signals within the given workstation.

15. The system of claim 14, further comprising means for signaling between the CEA units and the SMU sufficient to satisfy cache coherence operations

across the system when said cache coherence operations involve shared memory.

16. The system of claim 14, in which the SMU of said system includes a directory for keeping track of which workstation has which cache line.

17. The system of claim 14, in which said SMU cache directory includes means for keeping track of the ownership status of each workstation-owned cache line (READ SHARED, READ EXCLUSIVE, WRITE).

18. The system of claim 14, in which each CEA includes a directory of which workstation has which cache line.

19. The system of claim 14, in which said CEA cache directory includes means for keeping track of the ownership status of each workstation-owned cache line (READ SHARED, READ EXCLUSIVE, WRITE).

20. The system of claim 14, in which said CEA includes caching of shared-memory accesses.

21. A system, comprising:

multiple processing nodes and one or more shared memory nodes; each processing node containing some memory which is local and private to that node, and each shared memory node containing some other memory which is visible and usable by all nodes; and

a local cache system which provides caching for a first memory address region, and one or more shared-memory cache systems which provide caching for one or more other address regions.

22. A system, comprising multiple processing nodes; each processing node containing some memory which is local and private to that node, and a portion of other memory which is visible and usable by all nodes.

23. The system of claim 22, in which a local cache system provides the caching function for a first memory address region, and one or more shard-memory cache systems which provide the caching function for one or more other memory address regions.

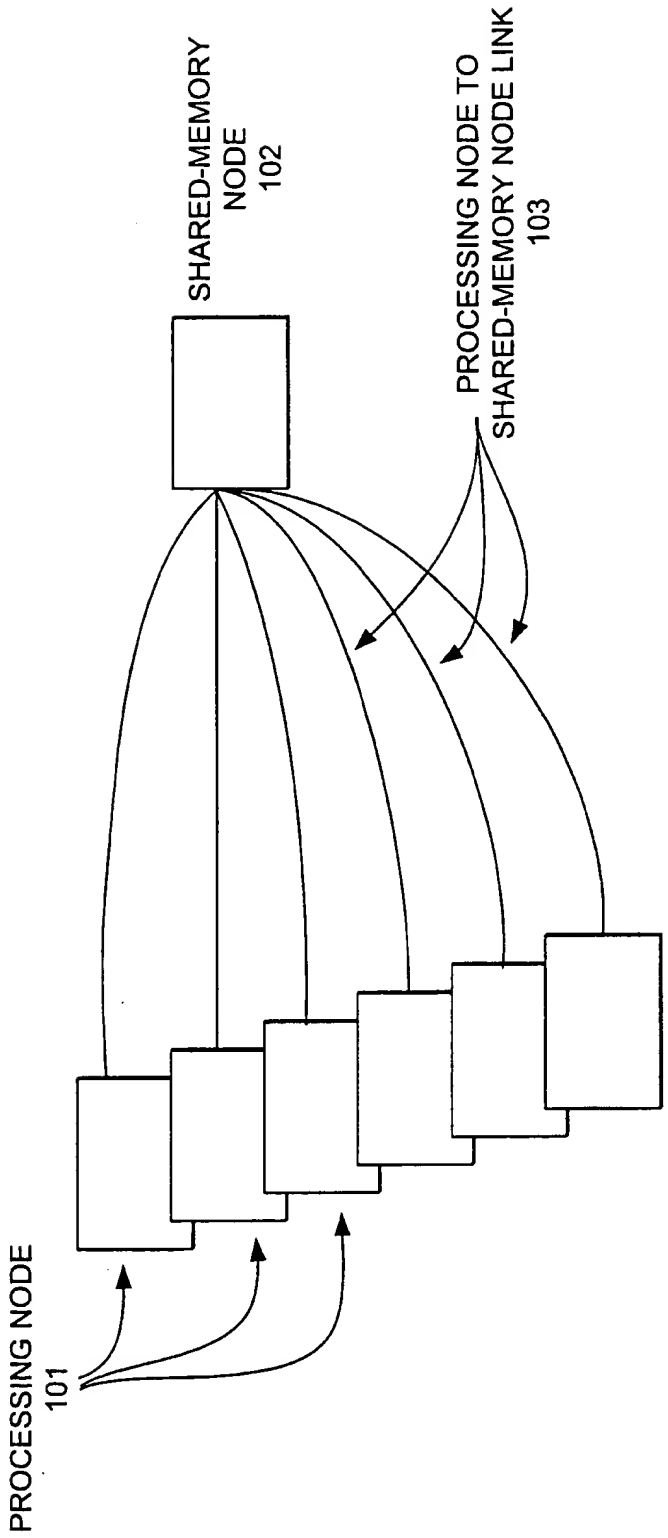
SHARED-MEMORY NODE 102
PROCESSING NODE TO SHARED-MEMORY NODE LINK 103
PROCESSING NODE 101

FIG. 1

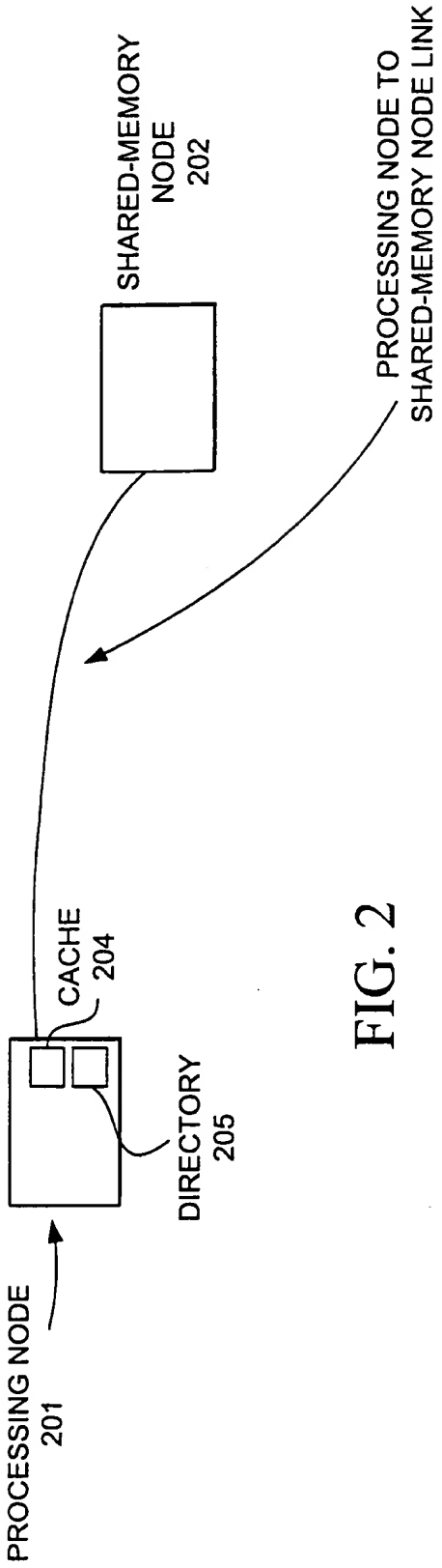
SHARED-MEMORY NODE 202
PROCESSING NODE TO SHARED-MEMORY NODE LINK 203
CACHE 204
DIRECTORY 205
PROCESSING NODE 201

FIG. 2

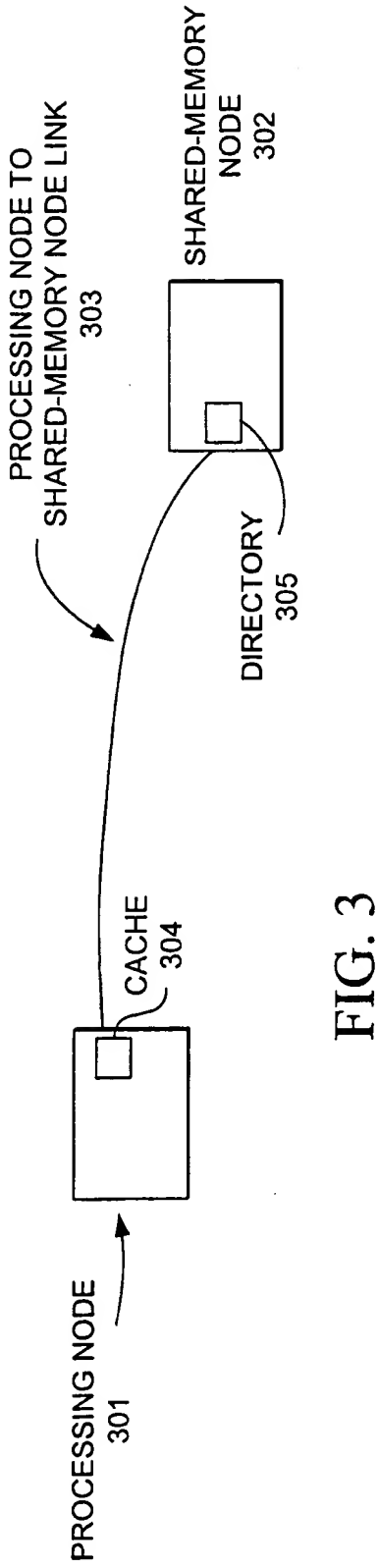
SHARED-MEMORY NODE 302
PROCESSING NODE TO SHARED-MEMORY NODE LINK 303
DIRECTORY 305
CACHE 304
PROCESSING NODE 301

FIG. 3

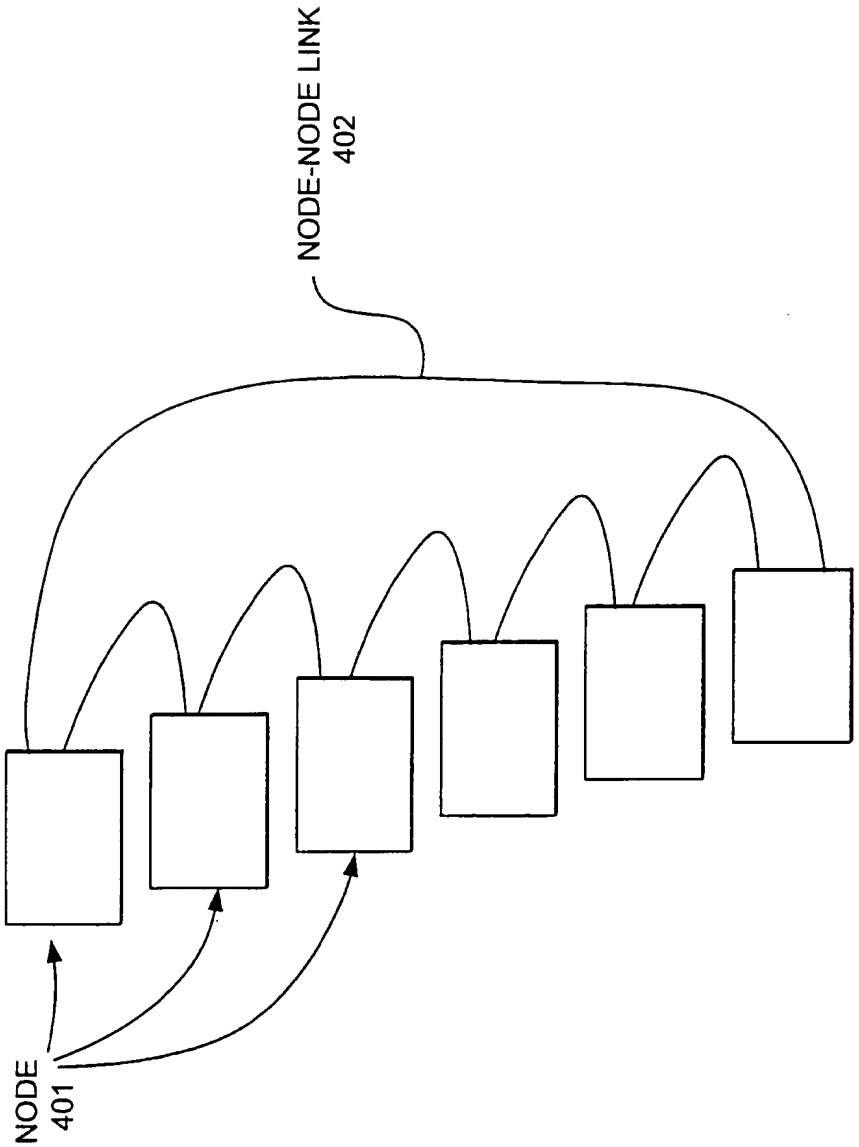

FIG. 4

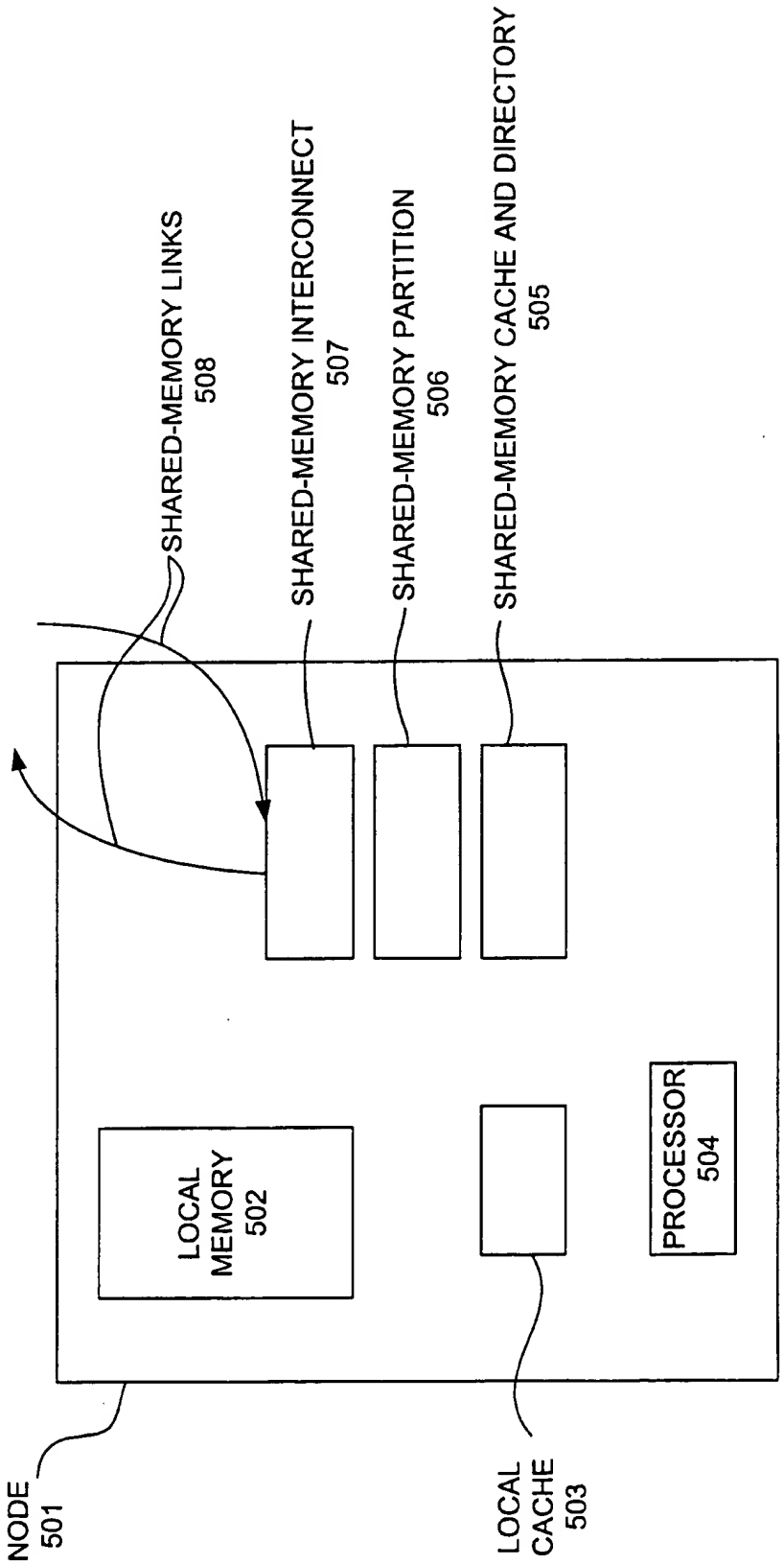
SHARED-MEMORY LINKS
508
SHARED-MEMORY INTERCONNECT
507
SHARED-MEMORY PARTITION
506
SHARED-MEMORY CACHE AND DIRECTORY
505
LOCAL MEMORY
502
PROCESSOR
504
NODE
501
LOCAL CACHE
503

FIG. 5

copy of the data and the interrogator reacts appropriately to that response. If a cache interrogation results in a Modified, then the cache has the only valid copy of the data and that copy is different from the copy in memory; the interrogator reacts appropriately to that response.

The interrogators and responses are also well understood in the art: there are two different kinds of interrogators: (1) the CPU needing access to data at that particular cache-line address; and (2) agents from other CPU's needing the data, after finding the data is not present in their local cache, searching all caches for the data. It should be mentioned that item (2) above is slightly simplified: sometimes an agent is generated for some other cause.

For a local CPU access, the interrogator action is rather simple if the data is present: get the data or modify the data. If the cache line is marked S and the CPU access is Store, then an agent is created to assure other caches are kept coherent. Similar results, well understood in the prior art, occur for other CPU actions and for the various agent actions, relative to each state in which a cache is for the addressed cache line.

Similarly, cache directories are well understood in the prior art. A standard example of a directory is one in which, if a cache line is not present in the local cache, the directory can be accessed for that cache-line address and will yield, via a bit pattern, which other caches may have copies of the cache line, the state in each of the other caches, and other similar information.

The present invention contemplates such an environment in which each free-standing computer in provided with very high-speed, low-latency communication means to a central shared-memory unit (SMU) which contains the memory shared among the collection of workstations which comprise the tight cluster. Each free-standing computer (e.g., workstation) is also provided with a specific interconnection to the SMU, said interface being the novel teaching of this invention.

One possible interface to the SMU would be a memory-alias interface. Such an interface would be to design an interface that appeared to the programmer (and to the processor) to be a memory adapter (usually called a memory card).